ANTON – A RULE-BASED COMPOSITION SYSTEM

Georg Boenn
Cardiff School of Creative & Cultural Industries
University of Glamorgan, UK

Martin Brain, Marina De Vos, John ffitch
Department of Computer Science
University of Bath, UK
jpff@cs.bath.ac.uk

ABSTRACT

We investigate the use of declarative logic programming in the automated composition of music. We show that it is possible to use Answer Set Programming (ASP) to create *ab initio* short musical pieces that are both melodic and harmonic and have an appropriate rhythmic structure based on Farey series.

Our system, ANTON, named in honour of our favourite composer of the second Viennese School, is presented as both a design and as a practical working system, showing that rule-based declarative systems can be used effectively. We report on our experience in using ASP in this system, and indicate a number of potentially exciting directions in which this system could develop, both musically and computationally.

1. INTRODUCTION

It has been said that deciding what the next note should be is the composer's fundamental problem! Many methods have been used both formally and informally to generate notes, from traditional techniques like repetition of all kinds, arpeggiation, inversion and the rest, to full algorithmic composition. As a side effect of other research in the use of logic in a range of problems, we started to investigate rules in music. We report here on the second version of our system that incorporates rules for melody, harmony and rhythm. Earlier versions of the system have been reported in the logic programming literature [6, 7, 8], but here we report for the first time the incorporation of rhythm for each part. We also introduce a methodology for evaluation of the musical results.

2. PROCESS-CONTROLLED COMPOSITION

Automation of compositional processes is not new. Perhaps the most frequently quoted is the Musical Dice Game (Musikalisches Würfelspiel) [10] often credited to Mozart, which was probably only one example of a game commonly played in Mozart's day.

More recent examples of the automation of compositional processes include serial techniques, pastiche via Markov chains [11] and use of chaotic processes [4]. This area is often called algorithmic composition, but that name is a little restrictive. In particular in this paper we describe not an algorithm but a set of rules, and the algorithm of realisation is not specified.

Our rule-based approach is closer to the constraint system of Anders [2], or the Bach chorale harmonisations of Ebcioğlu [12]. Moreover we are dedicated to writing compact and simple rules, with no sequentialisation or interpretation. Our work is based on the logic programming paradigm called Answer Set Programming.

3. ANSWER SET PROGRAMMING

Our system uses a form of logic programming called Answer Set Programming (ASP) [3] and specifically the language *AnsProlog*. Here we only present a short flavour of the language *AnsProlog*, and the interested reader is referred to [3] for a more in-depth coverage.

AnsProlog is a declarative knowledge representation languages that allows the programmer to describe a problem and the requirements on the solutions in an intuitive way, rather than describing the algorithm to find the solutions to the problem.

The basic components of the language are atoms, elements that can be assigned a truth value. An atom can be negated using *negation as failure* in order to create the *literal* not a. We say that not a is true if we cannot find evidence supporting the truth of a; somebody is not guilty unless evidence is available. If a is true then not a is false and vice versa.

Atoms and literals are used to create rules of the general form: $a \leftarrow B, \text{not } C$.where a is an atom, B and C are sets of atoms. Intuitively, this means *if all elements of B are known/true and no element of C is known/true, then a must be known/true.*, We refer to a as the head and $B \cup \text{not } C$ as the body of the rule. Rules with empty body are are called *facts*; the head should always be true. A *AnsProlog program* is a finite set of rules.

The semantics of *AnsProlog* are defined in terms of *answer sets* – assignments of true and false to all atoms in the program that satisfy the rules in a minimal and consistent fashion. A program has zero or more answer sets, each corresponding to a solution, or in this case, a different musical piece. We exploit this semantics to produce different musical pieces that do not break rules of the style.

When used as a programming language, *AnsProlog* is enhanced to contain constraints (e.g. $\leftarrow b, \text{not } c$) and choice rules (*e.g.* $\{a,b,c\} \leftarrow b, \text{not } c$). The former are

rules with an empty head, stating that an answer set cannot meet the conditions given in the body. The latter is a short hand notation for a conditional, non-deterministic choice; if the conditions in the body are met then one of the atoms in the head must be contained in an answer set. These additions are syntactic sugar and can be removed with linear, modular transformations (see [3]). Variables and predicated rules are also used and are handled, at the theoretical level and in most implementations, by instantiation (referred to as *grounding*).

An ASP system is composed of two processes, removing the variables from the program by instantiation with a *grounder* and computing answer sets of the propositional program with an *answer set solver*. We use GRINGO [14] and CLASP [13] for these.

4. ANTON

Our system implements the melody and counterpoint rules described by [16]. We treat these two components simultaneously; that is we do *not* create a melody and then harmonise it. It is also a composition system rather than improvisation, as it considers the whole piece out of time. Other modes of operation are supported, including computer aided composition and diagnosis of existing pieces.

The rules aim to compose in a sub-style of Renaissance Counterpoint, but there are also rules that govern sequence and timing. The rules can be considered in three areas; basic rules, style rules and rhythm rules.

4.1. Basic Rules

The basic rules define fundamental concepts of Western diatonic music such as consonant intervals, time passing, and valid note sequences, keys and modes. For example in the rule section called **notes** we define consonance:

```
%% Consonant intervals are
%% unison (0), minor third (3),
%% major third (4), fourth (5),
%% perfect fifth (7) (mod 12)
consonantInterval(0).
consonantInterval(3).
consonantInterval(4).
consonantInterval(5).
consonantInterval(7).
```

and rules for melodic minor, where T is a time step and P is a part; the rule can be read as *it is an error for a part to have move upwards to note 9 in the minor mode*, i.e. the minor sixth is only usable in a minor key when the melody is descending.

```
error(P,T,"Invalid in a minor key") :-
  chosenChromatic(P,T,9), upAt(P,T-1),
  mode(minor), partTime(P,T-1).
```

We use the predicate `error` rather than an empty clause so it can be instantiated to print a message when we are running in diagnostic mode (see section 6). When we are composing a constraint is used to prevent error occurring in the solution.

These rules are held in a number of files and are included via a PERL script to suit the requested style, in particular the mode, number of parts, *etc*.

The rules in this section apply in a wide range of styles.

4.2. Style Rules

The second class of rules relate to the details of Renaissance Counterpoint. For example we have the rules *A leap of an octave is only allowed from the fundamental*, and *No tri-tones*, the later encodes as:

```
error(MP,T,"Tri-tone") :-
  chosenNote(MP,T,N1),
  chosenNote(MP,T+2,N1+6).
error(MP,T,"Tri-tone") :-
  chosenNote(MP,T,N1),
  chosenNote(MP,T+2,N1-6).
```

These are negative rules, which are in effect constraints. In fact the rules are either definitional or constraints. For example the style rule that parts should not cross can easily be encoded as it should not being the case that a part is lower than the next part at any time. All the rules were developed from [16] in a very short time by a musician and a computer scientist sitting together. Indeed it is our contention that rule-based mechanisms are easy to develop.

4.3. Rhythm Rules

Our first version of ANTON created simple chorale-like pieces which is a major restriction, and so we have developed a set of rules that encode some types of rhythm. Much of this follows the use of Farey sequences as developed in [5], which can model a large variety of rhythmic structures.

Allen [1] describes the 13 possible relations between two temporal intervals. These have been used as the basis of temporal interval logics and give a very expressive language for talking about the relations (and thus harmonisation) between different parts. However directly expressing these relations within *AnsProlog* is problematic. Part of this difficulty due to current solvers requiring programs to be instantiated before answer sets can be computed. The 13 different relations over a large domain would result in an extremely large grounded program with which current solvers cannot cope. For this reason we have used a simplification of the Allen intervals, using a single predicate, `noteOverlap` which is true if there is any instant when both its arguments are extant. The `noteOverlap` predicate combines nine (begins, contained, overlaps, ends and their converses plus same) of the different Allen relations. This is are sufficient for the style of music that we are currently modelling; we will consider relaxing this in section 7.

The rhythm is created via a tree, where each node is a musical time interval and the children are equal spaced subdivisions. The number of children is limited by the

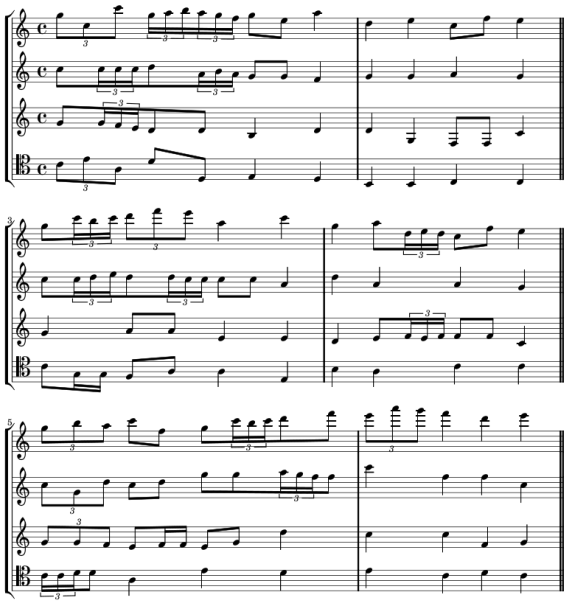

Figure 1. Fragment by ANTON2.0

order of the Farey sequence, and each child can be subdivided. While the code is written to allow any maximum order, for reasons of efficiency, and providing sufficient variety for our style of music, we restrict the order to be less than or equal to 3. This provides us with as a sufficiently rich subset of trees (and hence rhythms) for our musical genre.

The rhythm tree for our style has three duration levels (measure, metre and notes) with each its own set of rules, allowing us to specify points of stress, chords and time signatures. While for each part an individual note duration layer is created, all parts will have the same structure for the measure and metre layer. Without this constraint no consistency in piece can be guaranteed.

In figure 1 we present a small fragment of an ANTON piece, clearly showing the independence of the parts, and including triplets from the Farey sequence maximum of three.

4.4. Process Control

The complete system consists of three major phases; building the program, running the solver and interpreting the results. First we use a Perl script to assemble the rules required for a particular piece; for example suppose we wish to create a 4 bar piece in E major with rhythm one would use the Perl wrapper and write

```
$ programBuilder.pl --task=compose   \
                --mode=major --time=16 \
                --rhythm > program
```

which builds the *AnsProlog* program, giving the length and mode.

These rules are passed to the grounder and on to the solver to create solution; that is pieces of music that satisfy the rules provided.

```
$ gringo --compat < program          \
                | clasp > piece
```

The third stage is to interpret the results. At present we provide three interpretations; a printed list of notes, a Lilypond [15] score description, or a Csound [9] csd file incorporating a simple orchestra.

```
$ ./parse.pl --output=csound         \
                < piece > piece.csd
$ ./parse.pl --output=lilypond       \
                < piece > piece.ly
```

5. EVALUATION OF ANTON

An important question that is hard to answer is whether the notes created by ANTON are truly music. Clearly we have listened to many outputs as part of the development, and are satisfied with them. But in an attempt to give more credence to our assertions we have started a program of listening comparisons.[1] We have collected a number of ANTON pieces, together with transcriptions of historical composers and contemporary composers[2], and pieces written using random note selection.

A blind selection from these are then presented to a number of individuals, both musically trained and not, for rating.

At the time of writing we do not have sufficient trials for any statistical significance, but we can report that so far the results are as expected; ANTON does write music. We intend to develop this evaluation strategy in the near future.

6. OTHER CONSIDERATIONS

As described above this is a rule based composition system. But ANTON is much more than that. If a complete piece is presented to it it will indicate whether it conforms to the rules, diagnostic mode. By defining the error predicate to include messages as to which rule failed it is possible to see why the piece is not within the style. We envisage that this mode could be useful in testing student works that are supposed to be in a particular style. We did indeed use this mode while developing the rules.

There is also an intermediate state of use, when some of the piece is specified, and the system will provide the rest. In the simple case this becomes a harmoniser, such as might be for Bach chorales, but it could be the ending, or specific moments when certain notes or chords are required. In the longer term this might be of use in film music or similar commodity needs.

[1] But not currently completed

[2] Not necessarily of the same quality

It maybe possible to use an external program to implement the Allen relations in support of the ASP system, but this is a significant structural change.

There still remain a few areas that ANTON does not currently address. This include rhythmic structure, such as balancing, and the vexed problem of larger-scale design. We see the rôle of ANTON as more creating short passages, and a different set of rules will be needed to assemble them into a longer piece.

At present we have only one set of style rules. We have plans to explore other areas, and we are aware of an attempt to create trance. We are also exploring methods of varying the rule selection and presenting a more user-oriented interface to the underlying logic.

7. CONCLUSIONS

We have presented a rule-based system that can use recent advances in logic programming to create a constructive, diagnostic or flexible system for music in one particular style. The framework is adaptable to other musical styles with some additional sets of rules. The music it produces in compositional mode is generally pleasant and acceptable, according to very preliminary listener tests, and on occasions produces fine moments.

There are many ways the system could be developed. For example we might throw light on the compositional process by learning aspects of the rules, finding which are inconsistent or redundant, or determining the importance of rules. We could investigate whether there are 'unspoken' rules, and experiment to find unacknowledged rules of composition. One particularly interesting possibility is using the system to generate a large set of pieces, acquiring human evaluations of the 'quality' of each and then using techniques such as inductive logic programming to infer rules for composing 'good' pieces.

Real composers sometimes break the rules. This could be simulated by one of a number of extensions to answer set semantics (preferences, consistency restoring rules, defensible rules, *etc*.). However, how to systematise the knowledge of when it is acceptable to break the rules and in which contexts it is 'better' to break them is an open problem.

We present ANTON as a tool for better understanding of the compositional process.

8. REFERENCES

[1] J. F. Allen, "Maintaining Knowledge about Temporal Intervals," *CACM*, vol. 26, pp. 198–3, 1983.

[2] T. Anders, "Composing Music by Composing Rules: Design and Usage of a Generic Music Constraint System," Ph.D. dissertation, Queen's University, Belfast, Department of Music, 2007.

[3] C. Baral, *Knowledge Representation, Reasoning and Declarative Problem Solving*, 1st ed. Cambridge University Press, 2003.

[4] R. Bidlack, "Chaotic Systems as Simple (but Complex) Compositional Algorithms," *Computer Music Journal*, vol. 16, no. 3, pp. 33–47, Fall 1992.

[5] G. Boenn, "Automated Analysis and Transcription of Rhythm Data and their Use for Composition," Ph.D. dissertation, University of Bath, 2011.

[6] G. Boenn, M. Brain, M. De Vos, and J. ffitch, "Automatic Composition of Melodic and Harmonic Music by Answer Set Programming," in *International Conference on Logic Programming, ICLP08*, ser. Lecture Notes in Computer Science, vol. 4386. Springer Berlin / Heidelberg, 2008, pp. 160–174.

[7] ——, "ANTON: Composing Logic and Logic Composing," in *Logic Programming and Nonmonotonic Reasoning, 10th International Conference*, E. Erdem, F. Lin, and T. Schaub, Eds., Potsdam, Germany, September 2009, pp. 542–547.

[8] ——, "Automatic Music Composition using Answer Set Programming," *The Theory and Practise of Logic Programming*, 2010.

[9] R. Boulanger, Ed., *The Csound Book: Tutorials in Software Synthesis and Sound Design*. MIT Press, 2000.

[10] J. Chuang, "Mozart's Musikalisches Würfelspiel," http://sunsite.univie.ac.at/Mozart/dice/, 1995.

[11] D. Cope, "A Musical Learning Algorithm," *Computer Music Journal*, vol. 28, no. 3, pp. 12–27, Fall 2006.

[12] K. Ebcioğlu, "An Expert System for Harmonization of Chorales in the Style of J.S. Bach," Ph.D. dissertation, State University of New York, Buffalo, Department of Computer Science, 1986.

[13] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub, "Conflict-Driven Answer Set Solving," in *Proceeding of IJCAI07*, 2007, pp. 386–392.

[14] M. Gebser, T. Schaub, and S. Thiele, "GrinGo: A New Grounder for Answer Set Programming," in *Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*, ser. Lecture Notes in Artificial Intelligence, C. Baral, G. Brewka, and J. S. Schlipf, Eds., vol. 4483. Springer-Verlag, 2007, pp. 266–271.

[15] H.-W. Nienhuys and J. Nieuwenhuizen, "Lilypond, a system for automated music engraving," in *Proceedings of the XIV Colloquium on Musical Informatics (XIV CIM 2003)*, Firenze, Italy, May 2003.

[16] M. Thakar, *Counterpoint*. New Haven, 1990.